Vol. 14 No. 4 天 文 研 究 与 技 术 第 14 卷 第 4 期
Oct., 2017 ASTRONOMICAL RESEARCH AND TECHNOLOGY 2017 年 10 月

CN 53-1189/P ISSN 1672-7673

基于 Spark Streaming 的明安图射电频谱日像仪实时数据处理*

卫守林[1,2]，刘鹏翔[1]，王 锋[1,2]，邓 辉[1]，梁 波[1]，戴 伟[1,2]

（1. 昆明理工大学云南省计算机技术应用重点实验室，云南 昆明 650500；
2. 中国科学院云南天文台，云南 昆明 650011）

摘要：目前天文观测中对数据的实时处理需求越来越多，性能要求也越来越高，我国明安图射电频谱日像仪（MingantU SpEctral Radioheliograph，MUSER）是同时以高时间、高空间和高频率分辨率对太阳进行射电频谱成像的设备。在低频部分的日常观测中，包含了两方面的需求：（1）对历史数据的处理；（2）5 秒钟抽样观测数据的处理。抽样观测数据需要实时处理，并在监控终端显示，数据处理过程包含了数据校验、修正、成图、洁化等多个步骤，传统的单机处理模式已无法满足大数据量下的实时性要求。因此，实时数据计算中，使用 Spark Streaming 流式计算这一新兴的分布式计算方法，设计了自定义的接收器，并将多个图形处理器节点加入到分布式集群中。通过实验对性能进行评估，结果证明基于内存的高速执行引擎的特点能显著提高性能。期待能通过实验进一步优化算法和配置，获得更好的结果，并最终运用到实际环境中。

关键词：MUSER；射电天文；Spark；流式计算；实时计算
中图分类号：TP274⁺.2 **文献标识码**：A **文章编号**：1672-7673(2017)04-0421-08

大数据的分布式实时处理已在互联网应用中广泛使用，随着望远镜性能的大幅提升，在天文领域大数据的实时处理也有了越来越多的需求。位于我国内蒙古锡林郭勒盟的 400 MHz~15 GHz 厘米-分米波日像仪进入了试观测阶段[1]。在此阶段，试观测数据的正确、高效处理对设备的调试、误差修正有非常重要的作用，也能为未来常规观测的数据处理打下良好的基础。目前在低频部分的日常观测中，每分钟产生大约 1.92 GB 的原始观测数据写入磁盘，同时为了能够实时监测当前的设备状态和观测情况，日像仪每 5 秒产生抽样的观测数据，以 Socket 方式向外写出，也就是说需要在 5 s 内完成抽样观测数据的处理，并将处理结果发送到监控终端显示，否则就会产生数据积压，监控终端也就不能以准实时方式反映望远镜的观测情况。

数据的实时处理过程和原始数据处理过程一致，数据要经过预处理、天气数据标识、数据标定、校验、洁化、成图等多次迭代，成图、洁化等操作还需要在安装了图形处理器的机器上执行。呈现数据量大、数据来源多、类型多样、处理过程复杂等特点，这给数据的实时处理带来巨大挑战。使用互联网中广泛应用的开源分布式实时计算框架无疑能带来巨大的便利，不过实时的流式数据与互联网流式数据有很大不同：（1）次序可控，不同于互联网流式数据的出现不可预期；（2）二进制数据，不同于互联网的日志、点击信息等都是文本；（3）少量的统计逻辑；（4）无需保存原始数据；（5）结果数据量大于原始数据量。基于这些特点，选择合适的分布式实时处理框架将是实时数据处理成功的关键。

本文讨论使用分布式实时计算框架 Spark Streaming 处理实时数据，编写了实时数据接收和分发、处理模块，搭建了模拟环境对 Spark Streaming 从性能和可扩展方面进行测试分析和研究，实验结果表

* 基金项目：国家自然科学基金（11403009，U1231205）资助.
收稿日期：2017-02-24；修订日期：2017-03-20
作者简介：卫守林，男，硕士. 研究方向：天文技术与方法. Email：weishoulin@outlook.com

明，处理速度要快于实时数据产生的速度，满足了日像仪低频部分实时数据处理的要求。

1 实时数据流处理

分布式计算是解决大容量数据计算的，将大批量的计算任务切分成许多小任务，利用分布的计算节点计算每个小任务，最后将每个任务的计算结果合并得到最终的结果。扩展性、并行、分布式、节点同步、负载均衡、容错等是分布式计算最主要的特点[2]。分布式处理通常包含 3 种类型：(1)批量数据处理(Batch Data Processing)；(2)基于历史数据的交互式查询(Interactive Query)；(3)基于实时数据流的数据处理(Streaming Data Processing)[3]。

实时数据流处理或流式计算，是指数据或事件像水流的形式源源不断地到来，处理系统必须尽快对它们进行处理，最好是数据出现时便立刻对其进行处理，发生一个事件进行一次处理，而不是缓存起来成一批处理。批量数据处理通常读和写已归档的数据，而在数据流模型中，需要处理的输入数据(全部或部分)并不存储在可随机访问的磁盘或内存中，它们以一个或多个连续数据流的形式到达，并且带有时效性。批处理系统重视的是总数据处理的吞吐量，而实时计算关注数据处理的延迟，即希望进入的数据越快处理越好。

在调研分布式实时处理框架中，考虑过多个框架。首先是大名鼎鼎的 Apache Hadoop，Hadoop 是一个能够对大量数据进行分布式处理的软件框架，是目前最流行的大数据处理框架[4]。Hadoop 以一种可靠、高效、可伸缩的方式进行数据处理，它实现了并行与分布式计算 MapReduce 的编程思想。同时 Hadoop 不仅仅是一个框架，而是已经演变为一种分布式计算的生态圈，包含了多种计算框架，比如可伸缩的分布式迭代图处理系统 Giraph[5]，大规模的科学计算 Hama[6]，机器学习 Mahout [7]。通常将 Hadoop 归为批处理数据处理模型，处理的对象是历史数据，计算任务有开始的时间节点，数据处理完，任务结束并退出。而基于实时数据流处理的任务是永不退出的。现在也有一些组件，如 Micro-batchinMapReduce，Continuous MapReduce 使 Hadoop 支持流式数据处理。

S4 是 Yahoo 开源的分布式实时计算系统，主要是为了解决搜索广告的展现、处理用户的点击反馈[8]。Storm 是 Twitter 开源的分布式实时计算系统，可以简单、可靠地处理大量的数据流[9]。Storm 支持水平扩展，具有高容错性，且处理速度很快。消息通信基于 ZeroMQ，保证每个消息都会得到处理，几乎可以使用任意编程语言开发应用。在商业软件中，有 IBM 的 InfoShpere 和 EsperTech 的 Esper[10]。

Spark 是 UC Berkeley AMP Lab 开源的类 Hadoop MapReduce 的通用并行计算框架，Spark 基于 MapReduce 算法实现的分布式计算，采用 Scala 和 Java 语言实现，提供类似于 DryadLINQ 的集成语言编程接口，使用户可以非常容易地编写并行任务①。拥有 Hadoop MapReduce 具有的优点，但不同于 Hadoop 的是 Spark 的 Job 中间输出和结果可以保存在内存中，从而不再需要读写 HDFS，因此 Spark 能更好地适用于数据挖掘与机器学习等需要迭代的 MapReduce 算法。Spark 的核心是弹性分布式数据集，提供了比 MapReduce 更丰富的模型。

使用 Hadoop MapReduce 框架，虽然可以容易地实现较为复杂的处理和统计需求，但实时性却无法得到保证[2]。同时两个处理过程的中间数据，存储在 HDFS 中，这就造成磁盘文件的频繁读写，显著降低数据处理速度。通过 Spark 提供的应用程序编程接口和基于内存的高速执行引擎，用户可以结合使用流式、批处理和交互式查询应用[3]。Spark 的 Job 中间输出和结果可以保存在内存中，从而不再需要读写 HDFS，能显著提高处理速度。使用 Spark 作为执行引擎，具有高效和容错的特性，同时为实现复杂的算法提供和批处理类似的简单接口。Spark Streaming 基于微批量方式的计算和处理，可以用于处理实时的流数据[11]。Spark 能够支持 Python，Java 等多种编程语言，特别是 Python，有丰富的第三方应用库，如 numpy，pymatplot 和 scipy，这对天文数据处理尤为重要。这些优势都是选择 Spark Streaming 作为日像仪实时计算框架的原因。

① http://spark.apache.org/docs/latest/programming-guide.html

2　实时处理

明安图超宽频射电日像仪是采用综合孔径技术对太阳进行成像观测的射电望远镜，数据从相关器输出后，一般需要进行相位调整、条纹停止、数据标定、格式转换、成像、去卷积等[12]。在每个处理过程中还需要与其他系统交互，如天气气象信息的获取、仪器状态查询、光纤延迟数据查询、星表位置计算等。如图 1 显示了日像仪实时数据处理的流程图。

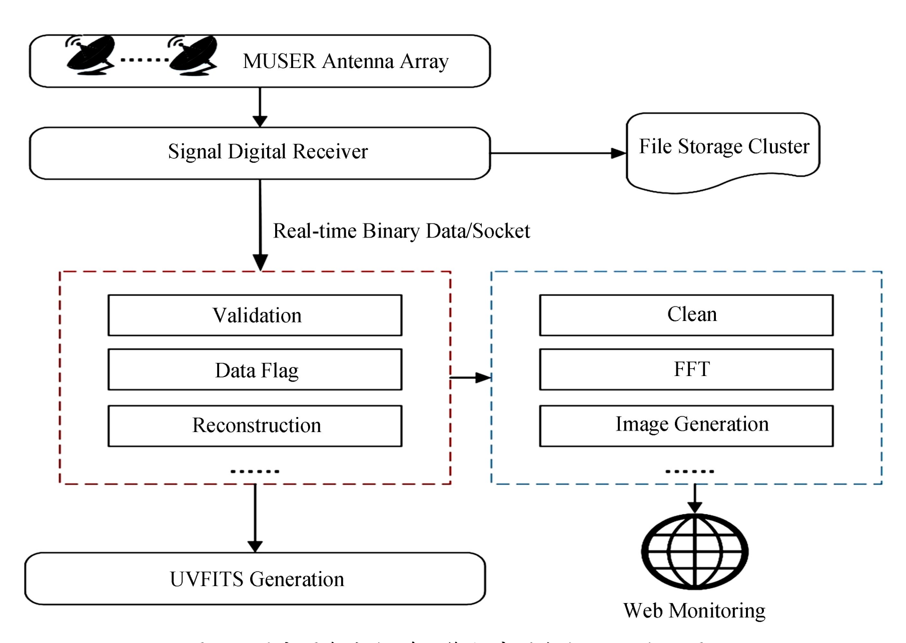


图 1　明安图射电频谱日像仪实时数据处理流程图

Fig. 1　The flowchart of real-time data processing in MUSER

由数字接收机产生的实时数据以 Socket/TCP 的方式向外写出数据，数据经过图 1 中左侧红色虚线框内的数据校验、重建等处理后生成 UVFITS 文件，再经过图 1 右侧蓝色虚线框内的处理后生成实时数据发布到网络监控端。其中蓝色框内的处理过程需要在安装了图形处理器的机器上执行。

综合孔径的射电频谱望远镜数据处理过程已有很成熟的理论和实践，但明安图射电频谱日像仪的数据处理有自己特殊的需求：

(1)实时性，日像仪将观测数据写入文件存储，同时产生实时数据流，实时数据流需要经过处理并将结果推送到监控端。这和传统的基于历史观测资料的数据处理有很大的不同。

(2)数据量大，日像仪的低频部分包含了 64 个通道，而高频部分包含了 528 个通道。图像处理部分的退卷积计算过程即使在图形处理器上操作仍非常耗时，低频部分的实时数据每 5 s 产生 8 帧数据，需要产生 128 个脏图，再加上洁化和成图的计算过程，这对实时性要求较高的实时发布是一个很大的挑战。

(3)过程复杂，如图 1 描述，实时数据在 UVFITS 生成和在图形处理器上的成图都需要多个处理步骤，计算过程复杂，同时要与多个系统交互协作完成。如数据校验过程中要考虑天气状态，就需要接入气象站的数据服务。

(4)异构的处理节点，集群中包含了普通的节点和带有图形处理器的节点，而图形处理等操作需要在图形处理器节点上执行。因此，分布式资源调度中，需要考虑不同类型的硬件资源调度。

综上，科学项目明安图射电频谱日像仪，于互联网应用，尽管存在相似的大数据处理需求，但在实时性、数据对象、处理过程上还存在很大的差异。因此，利用 Spark Streaming 的内存迭代计算特点，实现实时数据处理需考虑日像仪特殊的实时计算需求。

3　关键技术

3.1　自定义接收器

目前在低频部分的日常观测中，每 3 ms 产生 1 帧(100 000 Bytes)观测数据，每 5 s 产生 8 帧(8 × 100 000 Bytes)观测抽样数据。观测抽样数据需要实时处理，并将处理结果发送到监控终端显示。Spark Streaming 提供的 Python 接口，是加载文本对象的文件，或是连接 Flume，Kafka，HDFS 类型的特定存储系统，Spark Streaming 提供的例子也都是关于处理文本型的数据对象，而日像仪的实时数据是二进制的，且具有特定的数据格式(每 100 000 Bytes 大小为一帧)，因此需要自定义 Spark Streaming 的接收器，这个接收器能接收二进制数据并分析数据，转换成弹性分布式数据集(Resilient Distributed Dataset，RDD)。

Spark Streaming 支持从内置数据源包括 Flume、Kafka、Kinesis、HDFS、文件、套接字等加载数据，也可以自定义接收器从任意的流中接收数据。Spark 提供了 org.apache.spark.streaming.receiver.Receiver 这个抽象类，只需要继承这个抽象类，实现相关的方法，就可以实现自定义接收器，编程语言可以使用 Scala 或者 Java。

在 Receiver 抽象类中，有两个关键的方法需要重写：

(1) onStart()：这个函数主要负责在接收器启动时，做数据的接收工作；

(2) onStop()：这个函数主要负责在接收器停止时，做清理工作，停止接收线程。

不管是 onStart()和 onStop()方法都不可以无限期地阻塞。通常情况下，onStart()方法会启动一个新的线程负责接收数据，而 onStop()保证接收数据的线程被终止。接收数据的线程也可以使用 Receiver 类提供的 isStopped()方法检测是否可以停止接收数据。数据一旦被接收，这些数据可以通过调用 store(data)方法存储在 Spark 中，store(data)方法由 Receiver 类提供。store 方法是一个阻塞调用，只有当所有的数据都被存储到 Spark 里面才会返回。

自定义的接收器可以通过使用 streamingContext.receiverStream()方法在 Spark Streaming 应用程序中使用，但在 Python 的应用程序编程接口中，没有提供使用自定义接收器的方法，不能直接使用。因此需要一个中间的代理调用自定义的接收器，创建 MUSERStreamHelper，MUSERStreamHelper 是一个单例的类，提供一个静态方法，这个方法通过 Py4J 可以在 Python 环境中直接调用，如图 2，MUSERStreamHelper 和自定义的接收器 MUSERStreamReceiver 都使用 Java 编写，运行在 JVM 中。

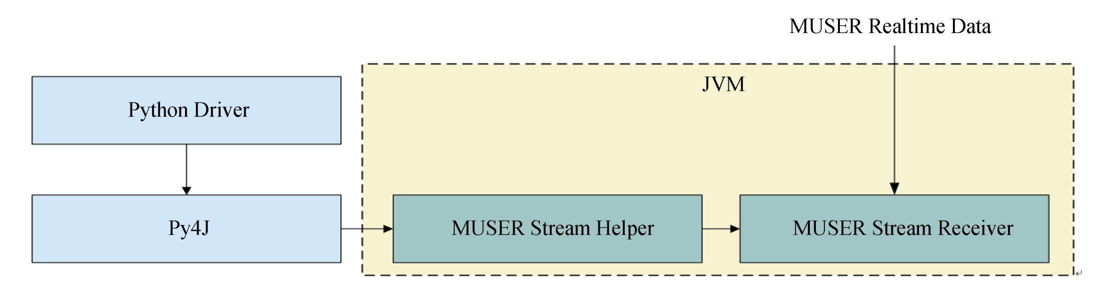


图 2　明安图射电频谱日像仪自定义 Receiver 实时数据处理流程图

Fig. 2　The flowchart of customized receiver for realtime binary data in MUSER

3.2　自定义分区

Spark 内部提供了 HashPartitioner 和 RangePartitioner 两种分区策略，HashPartitioner 是计算数据项的 Key 的 Hash 值，Hash 值相同的元素放入同一个分区；RangePartitioner 是将数据项的 Key 同一数据范围的数据放入同一分区。为了减少通信开销，尽量将需要进行相同操作的数据放在同一分区，另外为了提高集群的利用率，使弹性分布式数据集的各个分区上的数据量尽量均匀。

明安图射电频谱日像仪中使用观测时间和频段作为键值对弹性分布式数据集的 Key，使用

HashPartitioner 作为分区方式，因每个时间都不同，Hash 值也就不同，会造成分区过于分散，在做多个数据帧的积分操作时，会有较大的通信开销，使用 RangePartitioner 也不能完全满足要求。因此需要实现自定义的分区方式，可以使用 Spark 提供的 org.apache.spark.Partitioner 抽象类，继承 Partitioner 类并重写下面 3 种方法：

(1) numPartitions：Int：返回创建的分区数。

(2) getPartition(key：Any)：Int：返回给定键的分区编号(0 到 numPartitions-1)。

(3) equals()：判断相等性的标准方法。这个方法的实现非常重要，Spark 需要用这个方法检查分区器对象是否和其他分区器实例相同，这样 Spark 才可以判断两个弹性分布式数据集的分区方式是否相同。

在 Python 中，因为没有 Partitioner 抽象类，实现自定义的分区方式，不需要继承 Partitioner 类，只需要在 rdd.partitionBy() 里使用一个 Hash 函数，日像仪中使用观测时间和频段作为键值，格式为时间_频段，如：

20151101120854.354161_400

以下伪代码表示将 1 天 1 小时中的所有频段相同的放在一个分区中。

```
def customize_partition(key):
    keys = key.split("_")
    hashHour = hash(keys[0][6:8])  #计算小时的 Hash 值
    hashBand = hash(keys[1])  #计算频段的 Hash 值
    return hashHour + hashBand
rdd.partitionBy(4, customize_partition)
```

3.3　时间切片

Spark Streaming 流式数据处理的本质是将连续的数据持久化、离散化，然后进行批量处理。Spark Streaming 将输入的实时数据流以时间片为单位进行拆分，目前是以秒为单位拆分，然后以类似批处理的方式处理每个时间片的数据。互联网应用的实时流式数据由于数据流动态持续的特性，其数据项到达的次序与速度无法控制，并且随着时间的延续，数据流的体积在理论上是无限的。日像仪的实时数据是观测设备按观测时间顺序产生，其次序和速度(8 × 100 000 Bytes/5 s)确定。使用 Spark Streaming 的编写程序，首先需要声明 StreamingContext 对象，StreamingContext 对象的构造函数中有一个 batchDuration 参数，通过该参数定义 Spark Streaming 对数据流的切分间隔，batchDuration 参数显著影响数据处理速率，这个参数值可以通过检查端到端的延迟判断(可以在 Spark 驱动程序的日志中查看 Total delay 或者利用 StreamingListener 接口)。如果延迟维持稳定，那么系统稳定。如果延迟持续增长，那么系统无法跟上数据处理速率，是不稳定的。如果系统不稳定，除了可以适当减小 batchDuration 的值，同时也要考虑集群的处理能力，结合 Spark UI 任务的执行时间，找出延迟持续增长的原因。这里设置该值为 5，即 5 s。

实时数据流到达后，系统按照时间切片间隔将数据流切片后，返回 DStream，其本身封装了按时间片离散化了的数据流。DStream 中包含一个类型为 HashMap 成员变量 generatedRDDS，其中 Key 是时间片段，Value 就是弹性分布式数据集，DStream 操作和弹性分布式数据集的操作类似。

3.4　混合类型工作节点

由于图形处理器价格昂贵，不能在所有的计算节点上配置。这就要求数据处理一部分在普通的节点上计算，一部分在图形处理器节点上计算。Spark 集群运行模式，可以使用独立模式，另外可以结合使用 YARN 或 Mesos 资源调度器。但这 3 种模式下，都无法区分工作节点的类型，也就说 Spark 在分配计算任务时，没有办法区分哪些节点是图形处理器节点，也就无法正确地分配计算任务。为此将普通的节点和图形处理器节点分别单独部署在不同的集群中，如图 3，在普通的计算任务完成后，将计算结果发送到高速队列中，图形处理器节点再从队列中获取计算任务，队列在整个计算过程中充当了数据暂存的角色，数据量和处理性能要求不高，使用基于内存的自定义队列。

3.5 共享变量

共享变量是一种可以在 Spark 任务中使用的特殊类型的变量，Spark 中有两种类型的共享变量，广播变量和累加器。广播变量用来高效分发较大的对象，累加器用来对信息进行聚合。

日像仪的数据处理中，需要对高精度的观测目标的视位置进行相关计算，比如在每个相位差校正和生成 UVW 数据阶段。观测时为了确保相位补偿精度，需要观测目标的视位置计算精度优于 1 毫角秒，为此，采用精密的 JPL 星历表。使用 Spark 广播变量，将星历表高效地发送到所有的工作节点，提高星表查询和计算的速度。当广播一个比较大的值时，选择既快又好的序列化格式很重要，因为如果序列化对象的时间很长或者传送花费的时间太久，这段时间很容易成为性能瓶颈。

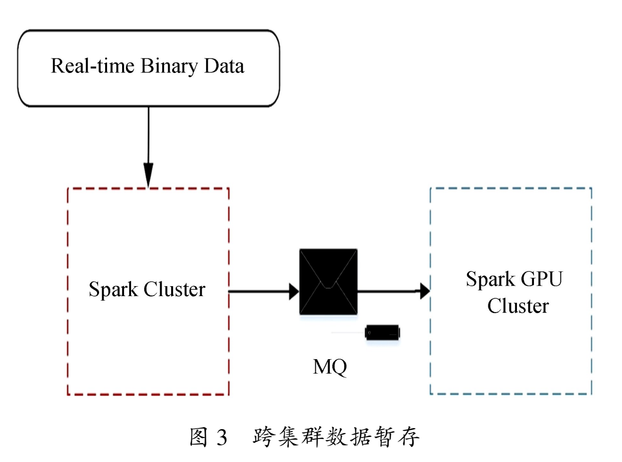


图 3 跨集群数据暂存

Fig. 3 The diagram of temporary data storage cross-clusters

JPL 星历表以二进制文件存储，频繁的文件打开和关闭是一个耗时的工作，如果能在多个数据元素间共享一次文件配置就比较高效。Spark 中使用基于分区对数据进行操作以避免为每个数据元素进行重复的配置，基于分区的 map(mapPartitions 函数)和 foreach(foreachPartitions 函数)，只对弹性分布式数据集的每个分区运行一次，这样可以帮助降低文件打开和关闭的频次，从而提高性能。

4 实 验

实验是在汉柏 PowerCube 创建的虚拟机上进行，使用 5 台配置相同的虚拟机，每台虚拟机配备了 16 G DDR2 内存，两路 Intel Xeon E5-2640 v2 CPUs，2.0 GHz，16 核中央处理器，16 G 主存，40 GB 硬盘。操作系统使用 CentOS7，Linux 内核版本 3.10.0。JDK 版本为 1.8，Spark 版本为 2.0.2，其中一个节点作为 Master 节点，同时也是 Worker 节点，其他 4 个节点作为 Worker 节点。每个 Worker 节点的 SPARK_WORKER_MEMORY 配置为 4 096 M，实验主要测试日像仪时间计算自定义的接收器、处理时间及整个系统的延迟，batchDuration 设置为 5 s。测试数据使用真实的原始观测数据，每 5 s 发送 8 个小帧，选取了大约 10 分钟(11:04:20 到 11:15:50)的测试结果如图 4。

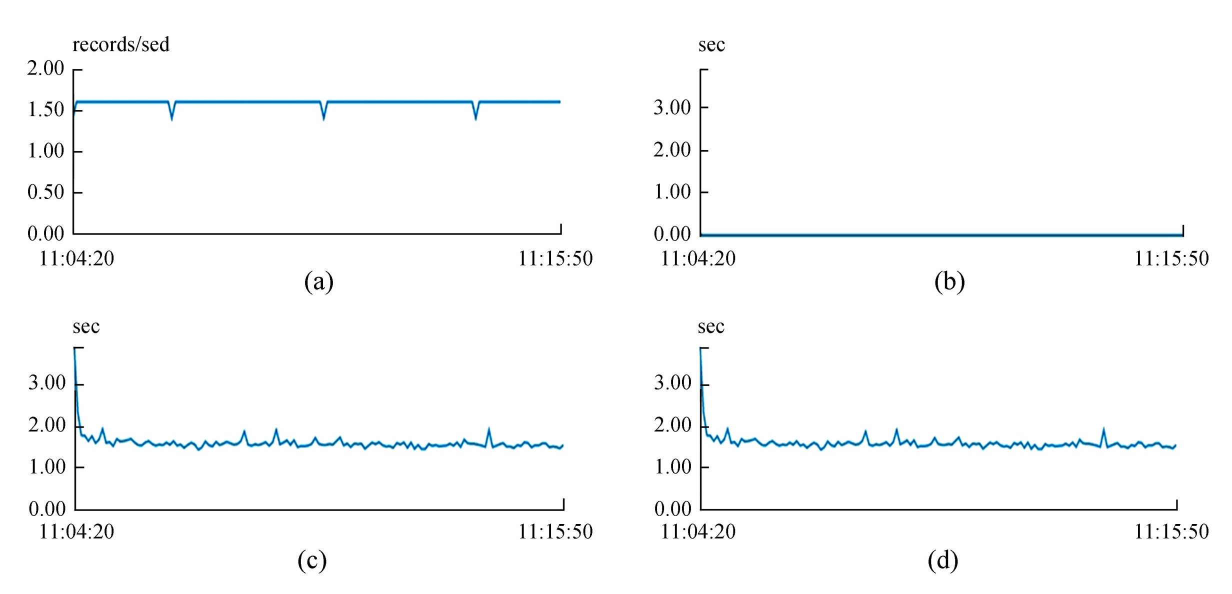


图 4 明安图射电频谱日像仪的 Spark 集群实时计算任务监控界面

Fig. 4 The monitoring interface of streaming job in MUSER's cluster

从图 4(a)数据接收的趋势图可以看出，接收器每秒接收的记录数和发送的记录数一致，并且系统保持稳定。图 4(b)显示了 Spark 的调度延迟在 2 ms 左右，可以忽略不计。从图 4(c)和图 4(d)数据处理时间随时间的变化图可以看出，在任务提交初期，由于驱动器和执行器的初始化延迟，在最初的任务执行时间较长，随后的任务处理时间比较稳定，大约在 1.6 s，满足日像仪实时任务处理的需求。

5 结束语

Spark Streaming 犹如其名“电光火石”，基于内存迭代计算的特点，显著提高了数据处理性能。目前，Spark Streaming 已在小范围使用，将数据以小批量方式处理后，可以同时兼容批处理和实时处理的算法，特别适用于某些同时处理历史数据和实时数据的业务场景，也适用于对实时性要求不是很高的实时计算业务，结合 Spark 提供的分布式数据处理能力，Spark Streaming 将在明安图超宽频射电日像仪和其他的天文数据处理中发挥更大的作用。但 Spark Streaming 作为一个全新的实时计算框架，还需要更多的业务场景，特别是在天文数据处理中验证其功能，另外不能很好地支持细粒度、异步的数据处理，性能还需要优化，稳定性也需要更长时间的考验。后续将继续研究 Spark Streaming 在天文方面的应用，并强化 Spark Streaming 的作业监控机制。

参考文献：

[1] 颜毅华，张坚，陈志军，等. 关于太阳厘米–分米波段频谱日像仪研究进展 [J]. 天文研究与技术——国家天文台台刊，2006，3(2)：91–98.
Yan Yihua，Zhang Jian，Chen Zhijun，et al. Progress on Chinese solar radioheliograph in cm-dm wavebands [J]. Astronomical Research & Technology——Publications of National Astronomical Observatories of China，2006，3(2)：91–98.

[2] 张建勋，古志民，郑超. 云计算研究进展综述 [J]. 计算机应用研究，2010，27(2)：429–433.
Zhang Jianxun，Gu Zhimin，Zheng Chao. Survey of research progress on cloud computing [J]. Application Research of Computers，2010，27(2)：429–433.

[3] 孙大为，张广艳，郑纬民. 大数据流式计算：关键技术及系统实例 [J]. 软件学报，2014，25(4)：839–862.
Sun Dawei，Zhang Guangyan，Zheng Weimin. Big data stream computing：technologies and instances [J]. Journal of Software，2014，25(4)：839–862.

[4] Dean J，Ghemawat S. MapReduce：simplified data processing on large clusters [J]. Communications of the ACM，2008，51(1)：107–113.

[5] Koschel A，Heine F，Astrova I，et al. Efficiency experiments on hadoop and giraph with PageRank [C] // 24th Euromicro International Conference on Parallel，Distributed，and Network-Based Processing. 2016：328–331.

[6] Seo S，Yoon E J，Kim J，et al. Hama：an efficient matrix computation with the mapreduce framework [C] // IEEE Second International Conference on Cloud Computing Technology and Science. 2010：721–726.

[7] Kathleen E，Pallickara S. On the performance of distributed clustering algorithms in file and streaming processing systems [C] // IEEE International Conference on Utility and Cloud Computing. 2011：33–40.

[8] Chauhan J, Chowdhury S A, Makaroff D. Performance evaluation of Yahoo! S4: afirst look [C]// 2012 Seventh International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC). 2012: 58-65.

[9] Samosir J, Indrawan-Santiago M, Haghighi P D. An evaluation of data stream processing systems for data driven applications [J]. Procedia Computer Science, 2016, 80: 439-449.

[10] Stoa S, Lindeberg M, Goebel V. Online analysis of myocardial ischemia from medical sensor data streams with esper [C]// 2008 ISABEL'08 First International Symposium on Applied Sciences on Biomedical and Communication Technologies. 2008: 1-5.

[11] Ghesmoune M, Lebbah M, Azzag H. Micro-Batching growing neural gas for clustering data streams using spark streaming [J]. Procedia Computer Science, 2015, 53(1): 158-166.

[12] Wang F, Mei Y, Deng H, et al. Distributed data-processing pipeline for Mingantu Ultrawide Spectral Radioheliograph [J]. Publications of the Astronomical Society of the Pacific, 2015, 127(950): 383-396.

Real-Time Data Processing in Mingantu Ultrawide Spectral Radio Heliograph Based on Spark Streaming

Wei Shoulin[1,2], Liu Pengxiang[1], Wang Feng[1,2], Deng Hui[1], Liang Bo[1], Dai Wei[1,2]

(1. Key Laboratory of Applications of Computer Technology of the Yunnan Provence, Kunming University of Science and Technology, Kunming 650500, China, Email: weishoulin@outlook.com;

2. Yunnan Observatories, Chinese Academy of Sciences, Kunming 650011, China)

Abstract: There is a growing demand for real-time processing in astronomical observations in recent years, meanwhile, the requirement for performance is also increasing dramatically. Mingantu Ultrawide Spectral Radio Heliograph (MUSER) is a synthetic aperture radio interferometer with high temporal, spatial and spectral resolution. In daily observation of low frequency, MUSER contains two aspects of data processing, historical data processing and sampling observational data which is produced every 5 seconds and processed in real-time mode. The procedure of raw data processing contains validation, correction, clean and other processing steps, then the results need to be transmitted in real-time mode to monitoring end without user constantly refreshing or sending a request. The traditional stand-alone processing mode has been unable to meet the requirements of large amounts of data in real-time mode. In this paper, we explored the use of Spark Streaming in a new approach for MUSER real-time calculations across multiple machines and evaluated its effectiveness and efficiency. A customized receiver was created for real-time binary stream of MUSER. We also extended the Spark cluster by adding multiple GPU's nodes. The experiments have shown that Spark Streaming can significantly improve MUSER real-time processing performance for its memory-based execution engine. We might look forward to optimize the algorithm through experiments and configurations so as to obtain better results, and apply it to the actual environment of MUSER finally.

Key words: MUSER; Radio astronomy; Spark; Streaming computing; Real-time computing